**Introduction**

Storing a piece of data in a PC hard drive is straightforward. However in other practical situations with a large scaling (e.g. RAID, clustered file system, network-attached/grid-oriented storage) it is usually needed to store files in a partitioned state, and put each of these data chunks into different disks/servers/etc. (This is called [data striping](https://en.wikipedia.org/wiki/Data_striping).)

Storing in the form of chunks has the extra advantage of not needing to store the same chunk twice on disk, thus saving disk space. The savings can be very impressive in cases of multiple highly-similar files.

As the guides and requirements, the proposed data storage needs to fulfill these four functionalities:

- loading of text files: When a file is loaded, it is divided into fixed sized chunks (e.g. 1K), the last chunk is padded such that it has the same size with the other chunks, the chunks are stored into your master storage unless they have not been stored before, and the filename to chunk relation is marked

- listing of loaded files: Your system should be able to list the names of the existing files

- retrieving back of text files: When a request to download a file is done, it should be generated back from its chunks and returned back to the user as it was uploaded

- deleting of text files: when a file delete is requested, the marks associating the file with the chunks and the chunks associated only with this file must be removed

**Add this into the data structure part, for time complexity analysis:**

Because dictionary lookup is O(1) assuming no collisions and traversing through the linked list is O(number of chunks) which equates to O(size of the file), we have:

Loading a text file: O(size of the file).

Listing of loaded files: O(nlog(n)), where n is the number of files in the storage. Listing itself is only O(n) but in our case an alphabetical order comparison sort is done before.

Retrieving back a file: O(size of the file).

Deleting a file: O(size of the file).

Obviously the above analysis is blind to actual interactions with Linux file systems. Considering now we are putting all of the chunks in the same directory, and if the disk is under ext4 file system (or similar ones that use [H-trees](https://ieeexplore.ieee.org/document/4351421) for file lookup), then the lookup time of the file system will be O(1), trivial compared to the functionalities. We also assume writing/deleting data on the disk is O(size of the file).

**Afterthoughts**

1 The number of the buckets

It is easy to see that there will be thousands of, if not more, chunk names in the chunkname dictionary. There are some solutions to the heightened chance of collisions, such as increasing the number of buckets, or adopting a dynamic resizing and rehashing scheme. Due to the uncertain nature of how many entries will actually be in the dictionary, maybe resizing/rehashing is the more appropriate solution. It will inevitably introduce time overheads, but estimating at O(times that the number of entries exceed upper limit) it is trivial.

If a better estimation of the usage scenario can be done (or an absolute limit of chunks in one directory), then a good size of the hash table can be picked.

The problem and proposed solutions are applicable to the filename dictionary as well.

2 Size of the chunk, granularity

The smaller the size of chunk, the more savings we can extract from saving the same chunk only once. However, not only it will make the dictionary size problem mentioned above worse, but also increase the storage overhead for storing chunk information. And because the hash function is O(1) regardless of the size of input, the constants in O(size of file) will increase as well.

Because the actual duplicate rate of files will vary greatly for different types of file and application, and because our data structures and algorithms are not sensitive to chunk size, it is possible to deploy a variable chunk size scheme for different files, based on usage analysis and/or possibly pattern recognition algorithms.